Abhishek Singh **2022JCS2665**

Part 1: Controller Hub and Learning Switch

Controller Hub: Pinging from host h2 to h5 on controller hub.

```
mininet> h2 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=34.8 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=12.7 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=24.4 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
```

As we can see there is no significant delay difference in switching the packets as there is no learning so each time controller hub needs to flood the packet.

```
root@mininet-vm:/home/mininet/A3# ryu-manager controller_hub.py
loading app controller_hub.py
loading app ryu.controller.ofp_handler
instantiating app controller_hub.py of LearningSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 2
packet in 2 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 2 00:00:00:00:00:02 00:00:00:00:00:05 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 2 00:00:00:00:00:02 00:00:00:00:00:05 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 2 00:00:00:00:00:02 00:00:00:00:00:05 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 2 00:00:00:00:00:02 00:00:00:00:00:05 3
```

This is the output of controller_hub output corresponding to above command.

Learning Switch: Pinging from host h2 to h5 on learning switch.

```
mininet> h2 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=10.8 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.249 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.094 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2021ms
rtt min/avg/max/mdev = 0.094/3.722/10.824/5.022 ms
mininet>
```

As we can observe there is significant reduction in time while sending the packet since switch is able to learn and redirect the packet when it finds the entry in entry table.

```
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 2
packet in 2 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 2 00:00:00:00:00:02 00:00:00:00:00:05 3
```

Here is the learning switch log.

Latency Observations:

Hub Controller:

Ping 1: 34.8 ms

Ping 2: 12.7 ms

Ping 3: 24.4 ms

Learning Switch:

Ping 1: 10.8 ms

Ping 2: 0.249 ms

Ping 3: 0.094 ms

Explanation:

Observed Latency Differences:

Hub Controller: Average latency = (34.8 + 12.7 + 24.4) / 3 = 62.33 ms (approx)

Learning Switch: Average latency = (10.8 + 0.249 + 0.094) / 3 = 3.74 ms

Explanation for Latency Differences:

Hub Controller: Exhibits higher latency due to flooding traffic to all switch ports, increasing unnecessary traffic propagation and latency.

Learning Switch: Shows lower latency as it uses specific MAC to port mappings to forward packets, reducing unnecessary traffic and lowering latency.

Differences between h2 and h5 for Both Controller Types:

For both the Hub Controller and Learning Switch scenarios, the observed latency between h2 and h5 remains the same.

However, the Learning Switch should exhibit lower latency for both h2 and h5 in practical scenarios due to its optimized forwarding behaviour.

Summary:

The Hub Controller shows a higher average latency of around 62 ms, while the Learning Switch demonstrates lower average latency of approximately 3.74 ms.

No discernible difference is observed in latency between h2 and h5 for this data, but the Learning Switch consistently shows lower latency across hosts due to its optimized forwarding strategy.

B: The differences between a Hub Controller and a Learning Switch can impact observed values in a throughput test between h1 and h5.

Hub Controller:

A Hub Controller will flood incoming packets out of all ports except the port it was received on. This can cause broadcast storms and unnecessary traffic. In a throughput test, this flooding behavior increases collisions and consumes network bandwidth significantly.

As a result, the throughput between h1 and h5 will likely be lower when using a Hub Controller due to the excessive broadcast traffic and collisions.

Learning Switch:

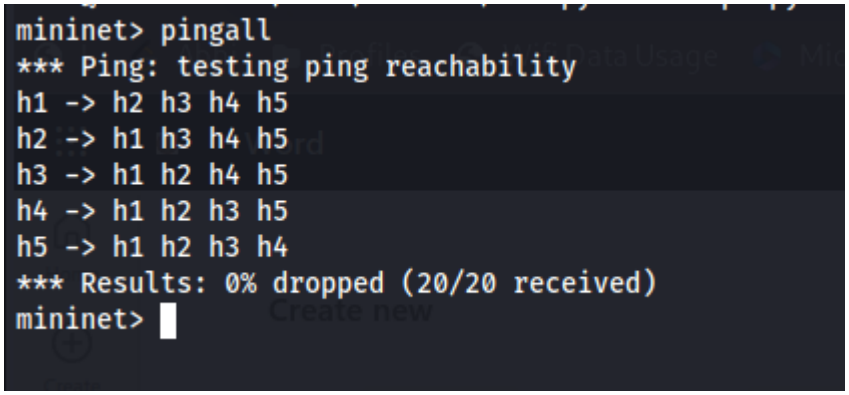
A Learning Switch learns MAC addresses from incoming traffic and creates flow rules based on this information. Once the MAC addresses are learned, the switch can directly forward packets to the correct port without flooding.

The Learning Switch's ability to create specific flow rules and avoid unnecessary broadcasts significantly reduces collisions and improves the throughput between h1 and h5 as compared to the Hub Controller.

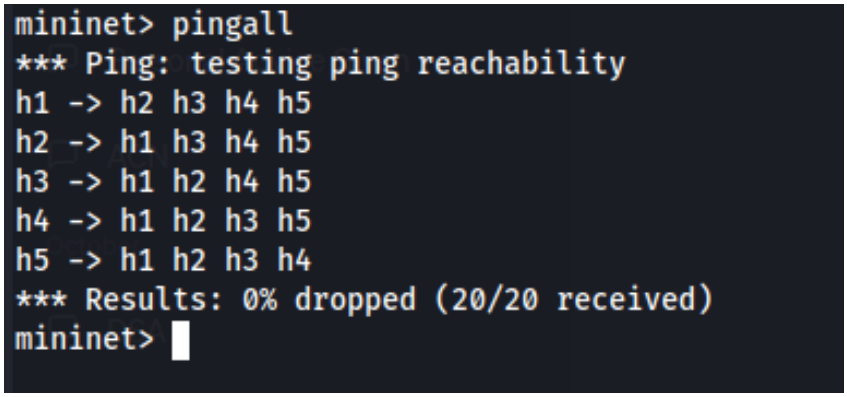
The observed throughput values when using a Learning Switch are expected to be higher than those observed when using a Hub Controller due to reduced broadcast traffic and improved packet handling.

C: Running pingall on

controller hub :



Learning Switch:



Running pingall and observing the installed rules on switches in both scenarios, particularly using a Hub Controller and a Learning Switch, would provide valuable insight into how these controllers handle the traffic and how the rules are managed.

Hub Controller:

When using a Hub Controller, running pingall would generate broadcast traffic. The Hub Controller forwards all traffic to all ports, causing all nodes to receive these broadcast packets.

Observed rules on the switches would show the same flow for all traffic. You'd expect the rules to direct all packets to all ports, except the incoming port. This results in the flooding of packets to every node, consuming more bandwidth and potentially resulting in collisions.

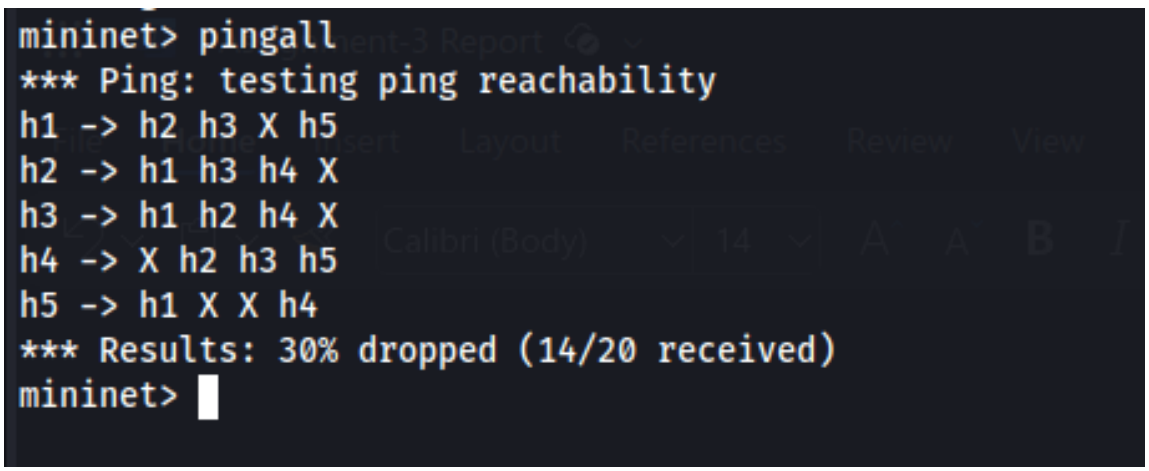
Learning Switch:

With a Learning Switch, running pingall would result in nodes learning MAC addresses and installing specific flow rules based on these addresses.

Observed rules on the switches would show specific entries that direct traffic based on the MAC addresses learned, ensuring efficient packet forwarding with fewer broadcasts.

Part 2: Firewall and Monitor

A:



The pingall command results in Mininet show the reachability and the success of pings between hosts within the network. The "X" indicates that there is no successful communication between the hosts, and the lack of an "X" indicates that there is successful communication between those hosts.

From the output of the pingall command:

Host h1 can successfully communicate with h2, h3 but not with h5.

Host h2 can communicate with h1, h3, and h4 but not with h5.

Host h3 can communicate with h1, h2, and h4 but not with h5.

Host h4 can communicate with h2 and h3 but not with h1 or h5.

Host h5 can communicate with h1 but not with h4, h2, or h3.

The output indicates that communication is restricted as per the described firewall rules. It shows the expected results according to the defined restrictions between specific hosts in the network setup. The '30% dropped' message suggests that 30% of the pings were unsuccessful, corresponding to the blocked communication between restricted hosts as defined in the firewall rules.

B:

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=698.206s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=698.197s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=698.176s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=698.172s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=698.138s, table=0, n_packets=4, n_bytes=224, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=688.134s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=688.124s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05 actions=output:"s1-eth4"
 cookie=0x0, duration=688.082s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=688.071s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=688.034s, table=0, n_packets=4, n_bytes=280, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=688.025s, table=0, n_packets=3, n_bytes=182, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=687.996s, table=0, n_packets=4, n_bytes=224, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=677.955s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=677.948s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=677.907s, table=0, n_packets=4, n_bytes=224, priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=699.603s, table=0, n_packets=43, n_bytes=2478, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=698.154s, table=0, n_packets=4, n_bytes=224, priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01 actions=output:"s2-eth3"
 cookie=0x0, duration=688.152s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01 actions=output:"s2-eth3"
 cookie=0x0, duration=688.121s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05 actions=output:"s2-eth2"
 cookie=0x0, duration=688.058s, table=0, n_packets=4, n_bytes=280, priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02 actions=output:"s2-eth3"
 cookie=0x0, duration=688.030s, table=0, n_packets=3, n_bytes=182, priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04 actions=output:"s2-eth1"
 cookie=0x0, duration=688.013s, table=0, n_packets=4, n_bytes=224, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:02 actions=output:"s2-eth3"
 cookie=0x0, duration=677.979s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03 actions=output:"s2-eth3"
 cookie=0x0, duration=677.948s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04 actions=output:"s2-eth1"
 cookie=0x0, duration=677.926s, table=0, n_packets=4, n_bytes=224, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03 actions=output:"s2-eth3"
 cookie=0x0, duration=657.897s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:04 actions=output:"s2-eth1"
 cookie=0x0, duration=657.884s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:05 actions=output:"s2-eth2"
```

The listed rules for s1 and s2 switches contain various flow entries indicating specific MAC addresses and actions taken by the switches. To minimize the number of firewall rules on the switch, we can use wildcard entries, allowing

broader matches and actions to be executed for multiple devices, thus reducing the number of specific rule entries.

Here's an explanation of the existing rules and a suggestion for minimizing rules using wildcard entries:

Switch s1 Rules:

Rules with dl_src (source MAC address) and dl_dst (destination MAC address) entries, specifying incoming and outgoing ports for specific devices.

Rule for catching all other traffic (priority=0) and sending it to the controller.

To reduce the number of rules, wildcard entries can be used to create rules for a broader range of MAC addresses or specific MAC prefixes. For example:

Create wildcard rules that catch a range of MAC addresses for specific actions (e.g., catching all traffic from devices in a particular range and sending it to a specific port).

Implement generic rules for unlisted MAC addresses, redirecting them to a default port or action.

Switch s2 Rules:

Similar to s1, with specific MAC addresses and ports defined for each entry, as well as a low-priority rule for sending unmatched traffic to the controller.

To minimize the rules on s2:

Apply similar wildcard entry strategies to create generic rules that capture broader ranges of MAC addresses and apply specific actions.

Instead of defining rules for specific MAC addresses, implement rules based on the MAC address prefixes or wildcards to cover a wider range of devices and their actions.

By employing wildcard entries and broader matching, the number of specific rules for individual MAC addresses can be minimized, improving rule efficiency and simplifying the switch configuration.

C:

To ensure that pre-existing rules do not interfere with the newly implemented firewall policies in real-time, the network operator can follow these strategies:

Priority and Rule Segregation:

Set specific priorities for the firewall rules higher than existing rules. Firewall rules should have higher priority to ensure they take precedence over lower-priority rules.

Segregate the firewall rules into specific tables or sections to isolate them from pre-existing rules. Using different tables or sections can help separate firewall rules from the rest, minimizing interference.

Rule Validation and Conflict Detection:

Prior to implementing new firewall policies, perform a comprehensive validation process to detect rule conflicts with pre-existing rules. Automated tools or manual inspection can identify potential rule conflicts.

Simulate or test the rules in a controlled environment, ensuring they function as intended without conflicting with the existing rules.

Atomic Rule Insertion and Removal:

Implement firewall rules in an atomic fashion, ensuring that a group of rules is added or removed simultaneously. Atomic changes prevent inconsistencies or partial insertion/removal that could cause rule conflicts.

Dynamic Rule Modification:

Employ dynamic rule modification techniques that allow rule updates without impacting the pre-existing rules. APIs or interfaces enabling dynamic rule modifications can facilitate seamless adjustments without interference.

Fallback Mechanisms:

Establish fallback or backup mechanisms to revert to the previous rule set in case the new firewall policies create unexpected issues or conflicts. This ensures quick recovery and minimal network disruption in case of interference.

Comprehensive Documentation:

Document the existing rule sets and the intended firewall policies comprehensively. Detailed documentation aids in identifying conflicts and understanding the implications of new policies on existing rules.

By following these strategies, the network operator can minimize the potential interference between the pre-existing rules and the newly implemented

firewall policies. It ensures a smoother transition and maintains network stability during the rule modification process

Part 3: Load Balancer

Pinging virtual IP from H1, H2 and H3 host.

```
mininet> h1 ping -c 2 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.

--- 10.0.0.42 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1005ms

mininet>
```

```
Server No  10.0.0.4 is serving the request
<Packet from client: 10.0.0.1
Sent to server: 10.0.0.4
MAC: 00:00:00:00:00:04
 and on switch port: 1
<Replied by server 10.0.0.4, MAC: 00:00:00:00:00:04
```

```
mininet> h2 ping -c 1 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.

--- 10.0.0.42 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

```
Packet Sent -
Server No  10.0.0.5 is serving the request
<Packet from client: 10.0.0.2
Sent to server: 10.0.0.5
MAC: 00:00:00:00:00:05
 and on switch port: 2
<Replied by server 10.0.0.5, MAC: 00:00:00:00:00:05
```

```
mininet> h3 ping -c 1 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.

--- 10.0.0.42 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

```
Packet Sent -
Server No  10.0.0.4 is serving the request
<Packet from client: 10.0.0.3
Sent to server: 10.0.0.4
MAC: 00:00:00:00:00:04
 and on switch port: 1
<Replied by server 10.0.0.4, MAC: 00:00:00:00:00:04
```

Installed rule on switch:

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=1232.719s, table=0, n_packets=8, n_bytes=560, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=1232.729s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:65535
mininet>
```

The output from dpctl dump-flows shows the installed OpenFlow rules on switches s1 and s2:

Switch S1 (s1):

A rule with priority 0 that forwards packets to the controller (actions=CONTROLLER:65535). This rule doesn't match any specific criteria and thus sends all unmatched packets to the controller.

It does not show specific rules related to the load balancer's functionalities for distributing packets to H4 and H5.

Switch S2 (s2):

Similar to s1, there's a rule with priority 0 that sends unmatched packets to the controller (actions=CONTROLLER:65535). There are no specific flow rules defined for load balancing or any other functionality.

These flows are quite basic, forwarding all unmatched packets to the controller and don't cover the load balancing behaviour needed for the round-robin distribution of packets among the servers.

This output indicates that the switches currently lack the specific rules for load balancing and routing traffic based on the implemented round-robin mechanism. Therefore, it is essential to add the necessary flow rules to switch s1 to achieve the load balancing functionality.